Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork

Samuel Barrett *
Kiva Systems
North Reading, MA 01864 USA
basamuel@kivasystems.com

Peter Stone
Dept. of Computer Science
The Univ. of Texas at Austin
Austin, TX 78712 USA
pstone@cs.utexas.edu

Abstract

Many scenarios require that robots work together as a team in order to effectively accomplish their tasks. However, pre-coordinating these teams may not always be possible given the growing number of companies and research labs creating these robots. Therefore, it is desirable for robots to be able to reason about *ad hoc teamwork* and adapt to new teammates on the fly. Past research on ad hoc teamwork has focused on relatively simple domains, but this paper demonstrates that agents can reason about ad hoc teamwork in complex scenarios. To handle these complex scenarios, we introduce a new algorithm, PLASTIC–Policy, that builds on an existing ad hoc teamwork approach. Specifically, PLASTIC–Policy learns policies to cooperate with past teammates and reuses these policies to quickly adapt to new teammates. This approach is tested in the 2D simulation soccer league of RoboCup using the half field offense task.

1 Introduction

As the presence of robots grows, so does their need to cooperate with one another. Usually, multiagent research focuses on the case where all of the robots have been given shared coordination protocols before they encounter each other (Grosz and Kraus 1996b; Tambe 1997; Horling et al. 1999). However, given the growing number of research laboratories and companies creating new robots, these robots may not all share a common coordination protocol. Therefore, it is desirable for these robots to be capable of learning and adapting to new teammates. This area of research is called *ad hoc teamwork* and focuses on the scenario in which the developers only design a single agent or small subset of agents that need to adapt to a variety of teammates (Stone et al. 2010).

Previous research into ad hoc teamwork has established a number of theoretical and empirical methods for handling unknown teammates, but these analyses largely focus on simple scenarios that may not be representative of the real problems that robots may encounter in ad hoc teams. This paper addresses this gap. The main contribution is the introduction of an algorithm (PLASTIC–Policy) for selecting effective actions for cooperating with a variety of teammates in complex domains. An additional contribution of this paper is the empirical evaluation of this algorithm in the RoboCup 2D simulation domain. The RoboCup 2D simulation league is a complex, multiagent domain in which teams of agents coordinate their actions in order to compete against other intelligent teams. The complexity arising from the dynamics of the domain and the opposing players makes this domain ideal for testing the scalability of our approach.

To quickly adapt to new teammates in complex scenarios, it is imperative to use knowledge learned about previous teammates. This paper introduces PLASTIC–Policy, which learns policies of how to cooperate with previous teammates and then reuses this knowledge to cooperate with new teammates. Learning these policies is treated as a reinforcement learning problem, where the inputs are features derived from the world state and the actions are soccer moves such as passing or dribbling. Then, PLASTIC–Policy observes the current teammates' behaviors to select which of these policies will lead to the best performance for the team.

The remainder of this paper is organized as follows. Section 2 situates this research in the literature, and then Section 3 describes the problem of ad hoc teamwork in more depth as well as presenting the background information. Section 4 introduces PLASTIC–Policy, the method used to tackle this problem. Section 5 presents the domain used in our empirical tests, and the results of these tests are given in Section 6. Finally, Section 7 concludes.

2 Related Work

Multiagent teamwork is a well studied area of research. Previous research into multiagent teams has largely focused on creating shared protocols for coordination and communication which will enable the team to cooperate effectively. For example, the STEAM (Tambe 1997) framework has agents build a partial hierarchy of joint actions. In this framework, agents monitor the progress of their plans and adapt their plans as conditions change while selectively using communication in order to stay coordinated. Alternatively, the Generalized Partial Global Planning (GPGP) (Decker and Lesser 1995) framework considers a library of coordination mechanisms from which to choose. Decker and Lesser argue that different coordination mechanisms are better suited to certain tasks and that a library of coordination mechanisms is

*This work was performed while Samuel was a graduate student at the University of Texas at Austin.

likely to perform better than a single monolithic mechanism. In SharedPlans (Grosz and Kraus 1996a), agents communicate their intents and beliefs in order to reason about coordinating joint actions, while revising these intents as conditions change.

While pre-coordinated multiagent teams are well studied, there has been less research into teams in which this pre-coordination is not available. This work builds on the ideas of Barrett et al. (2013), specifically learning about past teammates and using this knowledge to quickly adapt to new teammates. However, that work focused on a simpler domain in the form of a grid world, while this work investigates a complex, simulated robotics domain. As a result, while the overall approach from that work remains applicable, different learning methods are required. A piece of closely related work by MacAlpine et al. (2014) explores ad hoc teamwork in three RoboCup leagues. While that work presents a large scale test, it does not perform detailed analysis of learning-based ad hoc teamwork behaviors as we do in this paper.

One early exploration of ad hoc teamwork is that of Brafman and Tennenholtz (1996) in which a more experienced agent must teach its novice teammate in order to accomplish a repeated joint task. Another line of research, specifically in the RoboCup domain, was performed by Bowling and McCracken (2005). In their scenario, an agent has a different playbook from that of its teammates and must learn to adapt. Their agent evaluates each play over a large number of games and predicts the roles that its teammates will adopt. Other research into ad hoc teams includes Jones et al.'s (2006) work on pickup teams cooperating in a domain involving treasure hunts. Work into deciding which teammates should be selected to create the best ad hoc team was performed by Liemhetcharat and Veloso (2012). Additional work includes using stage games and biased adaptive play to cooperate with new teammates (Wu, Zilberstein, and Chen 2011). However, the majority of these works focus on simple domains and provide the agent with a great amount of expert knowledge.

A very closely related line of research is that of opponent modeling, in which agents reason about opponents rather than teammates. This area of research largely focuses on bounding the worst case scenario and often restricts its focus onto repeated matrix games. One interesting approach is the AWESOME algorithm (Conitzer and Sandholm 2007) which achieves convergence and rationality in repeated games. Valtazanos and Ramamoorthy explore modeling adversaries in the RoboCup domain in (2013). Another approach is to explicitly model and reason about other agents' beliefs such as the work on I-POMDPs (Gmytrasiewicz and Doshi 2005), I-DIDs (Doshi and Zeng 2009), and NIDs (Gal and Pfeffer 2008). However, modeling other agents' beliefs greatly expands the planning space, and these approaches do not currently scale to larger problems.

3 Problem Description and Background

This section describes the problem of ad hoc teamwork and then presents the mathematical model used to investigate the problem. It then presents the learning algorithm used as a subroutine in this paper.

3.1 Ad Hoc Teamwork

In this paper, we consider the scenario in which an ad hoc agent must cooperate with a team of agents that it has never seen before. If the agent is given coordination or communication protocols beforehand, it can easily cooperate with its teammates. However, in scenarios where this prior shared knowledge is not available, the agent should still be able to cooperate with a variety of its teammates. Specifically, an agent should be able to leverage its knowledge about previous teammates to help it cooperate with new teammates.

Evaluating an ad hoc team agent is difficult because the evaluation relies on how well the agents can cooperate with various teammates; we only create a single agent rather than an entire team. Therefore, to estimate the performance of agents in this setting, we adopt the evaluation framework proposed by Stone et al. (2010). In this work, Stone et al. formulate the performance of an ad hoc team agent as explicitly depending on the teammates and domains that it may encounter. We describe the domain and teammates used in our experiments in Section 5.

3.2 Mathematical model

Before specifying how to approach this ad hoc teamwork problem, it is helpful to describe how to model the problem. Specifically, we model the problem as a Markov Decision Process (MDP), which is a standard formalism in reinforcement learning (Sutton and Barto 1998) for describing an agent interacting with its environment. An MDP is 4-tuple (S, A, P, R), where S is a set of states, A is a set of actions, $P(s'|s,a)$ is the probability of transitioning from state s to s' when after taking action a, and $R(s,a)$ is a scalar reward given to the agent for taking action a in state s. In this framework, a policy π is a mapping from states to actions, which defines an agent's behavior for every state. The agent's goal is to find the policy that maximizes its long term expected rewards. For every state-action pair, $Q^*(s,a)$ represents the maximum long term reward that can be obtained from (s,a) and is defined by solving the Bellman equation

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

where $0 < \gamma < 1$ is the discount factor representing how much more immediate rewards are worth compared to delayed rewards. The optimal policy π^* is derived by choosing the action a that maximizes $Q^*(s,a)$ for every $s \in S$.

3.3 Learning a Policy

There are many algorithms for learning policies in an MDP. In this work, our agent uses the Fitted Q Iteration (FQI) algorithm introduced by Ernst et al. (2005) Similar to Value Iteration (VI), FQI iteratively backs up rewards to improve its estimates of the values of states. Rather than looking at every state and every possible outcome from each state, FQI uses samples of these states and outcomes to approximate the values of state-action pairs. This approximation allows FQI to find solutions for complex, continuous domains. Alternative policy learning algorithms can be used, such as Q-learning (Watkins 1989) or policy search (Deisenroth, Neumann, and Peters 2013).

To collect samples of the domain, the agent first performs a number of exploratory actions. From each action, the agent stores the tuple $\langle s, a, r, s' \rangle$, where s is the original state, a is the action, r is the reward, and s' is the resulting state. An advantage of the FQI algorithm is that this data can be collected in parallel from a number of tests. At each iteration, the agent updates the following equation for each tuple

$$Q(s, a) = r + \gamma * \max_{a'} Q(s', a')$$

where $Q(s, a)$ is initialized to 0. Q is an estimate of the optimal value function, Q^*, and this estimate is iteratively improved by looping over the stored samples. To handle continuous state spaces, Q is not stored exactly in a table; instead, its value is approximated using function approximation. In this paper, the continuous state features are converted into a set of binary features using CMAC tile-coding (Albus 1971; 1975), and the estimate of $Q(s, a)$ is given by

$$\hat{Q}(s, a) = \sum_i w_i f_i$$

where f_i is the i^{th} binary feature and w_i is the weight given to the feature with updates split uniformly between the active features.

4 PLASTIC–Policy

Intelligent behaviors for cooperating with teammates can be difficult to design, even if the teammates' behaviors are known. In complex domains, there are the added difficulties that the actions may be noisy and the sequences of actions required to accomplish the task may be long. Therefore, this section introduces Planning and Learning to Adapt Swiftly to Teammates to Improve Cooperation – Policy (PLASTIC–Policy), an algorithm that learns effective policies for cooperating with different teammates and then selects between these policies when encountering unknown teammates. This section describes PLASTIC–Policy, shown in Algorithm 1.

PLASTIC–Policy is an extension of the algorithm proposed by Barrett et al. (2013). Compared to that work, the main differences are 1) the use of a policy-based, model-free method rather than a model-based approach and 2) the evaluations are on the much more complex domain of HFO. Barrett et al.'s model-based approach works in domains with small, discrete state spaces, but cannot scale to a domain such as HFO. PLASTIC–Policy scales independently of the size of the state space given the learned policies.

4.1 Learning about Teammates

In complex domains, the MDP's transition function can be problematic to learn and represent. Therefore, rather than explicitly modeling the MDP's transition function, the agent directly uses samples taken from tasks with its teammates. To learn the policy (Line 4), PLASTIC–Policy employs the FQI algorithm presented in Section 3.3. For each set of teammates, the agent plays in a number of exploratory episodes in which it explores the different available actions. From each action, the agent stores the tuple $\langle s, a, r, s' \rangle$, where s is the original state, a is the action, r is the reward, and s' is the resulting state. Then, FQI is performed over these stored experiences, resulting in a policy that allows the agent to cooperate with that type of teammates. Note that a policy is learned for each possible teammate type.

Algorithm 1 Pseudocode of PLASTIC–Policy

```
 1: function PLASTIC–Policy:
        inputs:
            PriorT       ▷ previously encountered teammates
            BehPrior     ▷ prior distr over teammate types
    ▷ learn policies for prior teammates
 2:     Knowl = {}
 3:     for $t \in$ PriorT do
 4:         Learn policy $\pi$ for cooperating with $t$
 5:         Learn nearest-neighbor model $m$ of $t$
 6:         Knowl = Knowl $\cup \{(\pi, m)\}$
 7:     BehDistr = BehPrior(Knowl)

    ▷ act in the domain
 8:     Initialize $s$
 9:     while $s$ is not terminal do
10:         $(\pi, m) = \mathrm{argmax}$ BehDistr
11:         $a = \pi(s)$
12:         Take action $a$ and observe $r, s'$
13:         BehDistr = UpdateBeliefs(BehDistr, $s, a$)

14: function UpdateBeliefs(BehDistr, $s, a$):
15:     for $(\pi, m) \in$ BehDistr do
16:         loss = $1 - P(a|m, s)$
17:         BehDistr($m$)$* = (1 - \eta$loss)
18:     Normalize BehDistr
19:     return BehDistr
```

These learned policies specify how to cooperate with a teammate, but not how that teammate may act. To determine the type of teammate the agent is cooperating with, PLASTIC–Policy also learns an approximate model of the teammates in Line 5. Using the same experiences collected for learning the policy, PLASTIC–Policy builds a nearest neighbor model of the teammates' behavior, mapping states s to the teammates' next states s'.

4.2 Policy Selection

When an agent joins a new team, it must decide how to act with these teammates. If it has copious amounts of time, it can learn a policy for cooperating with these teammates. However, if its time is limited, it must adapt more efficiently.

We assume that the agent has previously played with a number of different teams, and the agent learns a policy for each of these teams. When it joins a new team, the agent can then reuse the knowledge it has learned from these teams to adapt more quickly to the new team. One way of reusing this knowledge is to select from these learned policies. If the agent knows its teammates' identities and has previously played with this team, the agent can directly used the learned policy. However, if it does not know their identities, the agent must select from a set of learned policies.

Many similar decision making problems can be modeled as multi-armed bandit problems when the problem is stateless. In this setting, selecting an arm corresponds to playing one of the learned policies for an episode. Over time, the agent can estimate the expected chance of scoring of each policy by selecting that policy a number of times and observing the binary outcome of scoring versus not scoring.

However, this approach may require a large number of trials as the outcomes of playing each policy is very noisy based on the complexity of the domain. To learn more quickly, PLASTIC–Policy maintains the probability of the new team being similar to a previously observed team, described in Lines 14–19. One option to update these probabilities is to observe the actions the team performs and use Bayes' theorem. However, Bayes' theorem may drop the posterior probability of a similar team to 0 for a single wrong prediction. Therefore, PLASTIC–Policy uses the approach of Barrett et al. (2013) of updating these probabilities using the polynomial weights algorithm from regret minimization (Blum and Mansour 2007), shown on Lines 16–17, where η is empirically chosen to be 0.1.

To update the probability of a teammate type, PLASTIC–Policy uses the corresponding model m, learned in Line 5. The model m maps a state s to the team's next state s'. For each old team, PLASTIC–Policy finds the stored state $\hat{s}$ closest to s and its next state $\hat{s}'$. Then, for each component of the state, it computes the difference between s' and $\hat{s}'$. PLASTIC–Policy assumes that the MDP's noise is normal and calculates the probability that each difference was drawn from the noise distribution. The product of these factors is the point estimate of the probability of the previous team taking the observed action.

Given these beliefs, PLASTIC–Policy chooses the most likely teammate type in Line 10 and acts following the corresponding policy in Lines 11–12.

5 Domain Description

The 2D Simulation League is one of the oldest leagues in RoboCup and is therefore one of the best studied, both in competition and in research. In this domain, teams of 11 autonomous agents play soccer on a simulated 2D field for two 5 minute halves. The game lasts for 6,000 simulation steps, each lasting 100 ms. At each step, these agents receive noisy sensory information such as their location, the location of the ball, and the locations of nearby agents. After processing this information, agents select abstract actions that describe how they move in the world, such as dashing, kicking, and turning. This domain is chosen to provide a testbed for teamwork in a complex domain without requiring focus on areas such as computer vision and legged locomotion.

5.1 Half-Field Offense

Rather than use full 10 minute 11 on 11 game, this work instead uses the quicker task of half field offense introduced by Kalyanakrishnan et al. (2007). In Half Field Offense (HFO), a set of offensive agents attempt to score on a set of defensive agents, including a goalie, without letting the defense capture the ball. A view of this game is shown in Figure 1. This task is useful as it allows for much faster evaluation of team performance than running full games as well as providing a simpler domain in which to focus on improving ball control. In this work, we use two variations: 1) the limited version with two offensive players attempting to score on two defenders (including the goalie) and 2) the full version with four attackers attempting to score on five defenders. We consider two versions of the problem to analyze the scalability of the approaches.

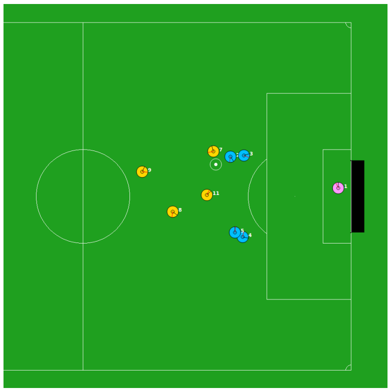

Figure 1: A screenshot of half field offense in the 2D soccer simulation league. The yellow agent number 11 is under our control, and remaining yellow players are its externally created teammates. These agents are trying to score against the blue defenders.

If the offensive team scores a goal, they win. If the ball leaves the offensive half of the field, the defense captures the ball, or no goal is scored within 500 simulation steps (50 seconds), then the offensive team loses.

At the beginning of each episode, the ball is moved to a random location within the 25% of the offensive half closest to the midline. Let *length* be the length of the soccer pitch. Offensive players start on randomly selected vertices forming a square around the ball with edge length $0.2 \cdot$ *length* with an added offset uniformly randomly selected in $[0, 0.1 \cdot$ *length*$]$. The goalie begins in the center of the goal, and the remaining defensive players start randomly in the back half of their defensive half.

In order to run some existing teams used in the RoboCup competition, it is necessary to field the entire 11 player team for the agents to behave correctly. Therefore, it is necessary to create the entire team and then constrain the additional players to stay away from play, only using the agents needed for half field offense. We use this approach to control the number of agents without access to the other agent's source code. We choose a fixed set of player numbers for the teammates, based on which player numbers tended to play offensive positions in observed play. The defensive players use the behavior created by Helios in the limited version of HFO. In the full HFO, the defense uses the agent2d behavior provided in the code release by Helios (Akiyama 2010). All past research on HFO has focused on creating full teams that are pre-coordinated. This work is the first to study ad hoc teamwork in this setting.

5.2 Teammates

In ad hoc teamwork research, it is important to use *externally-created* teammates to evaluate the various ad hoc team agents. Externally-created teammates are created by developers other than the authors and represent real agents that are created for the domain when developers do not plan for ad hoc teamwork scenarios. They are useful because their development is not biased to make them more likely to cooperate with ad hoc team agents. As part of the 2D simulation league competition, teams are required to release binary versions of their agents following the competition. Specifically, we use the binary releases from the 2013 competition. These agents provide an excellent source of *externally-created* teammates with which to test the possible ad hoc team agents. Specifically, we use 6 of the top 8 teams from the 2013 competition, omitting 2 as they do not support playing games faster than real time. In addition, we use the agent2d team. Therefore, there are a total of 7 possible teams that our agent may encounter: agent2d, aut, axiom, cyrus, gliders, helios, and yushan.

5.3 Grounding the Model

This section describes how we model the HFO domain as an MDP.

State A state $s \in S$ describes the current positions, orientations, and velocities of the agents as well as the position and velocity of the ball. In this work, we use the noiseless versions of these values to permit for simpler learning. However, the teammates' and opponents' behaviors have significant noise. As a result adding environmental noise would not qualitatively change the complexity of the domain.

Actions In the 2D simulation league, agents act by selecting whether to dash, turn, or kick and specify values such as the power and angle to kick at. Combining these actions to accomplish the desired results is a difficult problem. Therefore, this work builds on the code release by Helios (Akiyama 2010). This code release provides a number of high level actions, such as passing, shooting, or moving to a specified point.

We use 6 high level actions when the agent has the ball:

1. Shoot – shoot the ball at the goal, avoiding any opponents
2. Short dribble – dribble the ball while maintaining control
3. Long dribble – kick ball and chase it
4. Pass$_0$ – pass to teammate 0
5. Pass$_1$ – pass to teammate 1
6. Pass$_2$ – pass to teammate 2

Each action considers a number of possible movements of the ball and evaluates their effectiveness given the locations of the agent's opponents and teammates. Each action therefore represents a number of possible actions that are reduced to discrete actions using the agent2d evaluation function. While using these high level actions restricts the possibilities that the agent can take, it also enables the agent to learn more quickly and prune out ineffective actions, allowing it to select more intelligent actions with fewer samples.

Additionally, the agent can select how it moves when it is away from the ball. As the agent can take a continuous turn action or a continuous dash action every time step, it is helpful to again use a set of high level actions, in this case 7:

1. Stay in the current position
2. Move towards the ball
3. Move towards the opposing goal
4. Move towards the nearest teammate
5. Move away from the nearest teammate
6. Move towards the nearest opponent
7. Move away from the nearest opponent

These actions provide the agent a number of possible actions that adapt to its changing environment, while constraining the number of possible actions.

Transition Function The transition function is defined by a combination of the simulated physics of the domain as well as the actions selected by the other agents. The agent does not directly model this function; instead, it stores samples observed from played games as described in Section 4.1.

Reward Function The reward function is 1,000 when the offense wins, -1,000 when the defense wins, and -1 per each time step taken in the episode. The value of 1,000 is chosen to be greater than the effects of step rewards over the whole episode, but not so great as to completely outweigh these effects. Other values were tested with similar results.

5.4 Collecting Data in HFO

We treat an action as going from when an agent has possession of the ball until the action ends, another agent holds the ball, or the episode has ended. Given that we only control a single agent, the teammates follow their own policies. The agent collects data about its actions and those of its teammates' in the form $\langle s, a, r, s' \rangle$ where the a is our agent's actions. The agent does not directly store the actions of its teammates, instead storing the resulting world states, which include the effects of its teammates' actions. If we controlled all of the agents, we would also consider the action from the teammates' perspectives. The agent observes 100,000 episodes of HFO with each type of teammates. These episodes contain the agent's actions when the agent has the ball as well as when it is away from the ball.

There are many ways to represent the state of a game of half field offense. Ideally, we want a compact representation that allows the agent to learn quickly by generalizing its knowledge about a state to similar states without over-constraining the policy. Therefore, we select 20 features given that there are 3 teammates:

- X position – the agent's x position on the field
- Y position – the agent's y position on the field
- Orientation – the direction that the agent is facing
- Goal opening angle – the size of the largest open angle of the agent to the goal, shown as θ_g in Figure 2
- Teammate i's goal opening angle – the teammate's goal opening angle
- Distance to opponent – distance to the closest opponent
- Distance from teammate i to opponent – the distance from the teammate to the closest opponent
- Pass opening angle i – the open angle available to pass to the teammate, shown as θ_p in Figure 2

6 Results

This section presents the results of PLASTIC–Policy in HFO. Results are averaged over 1,000 trials, each consisting of a series of games of half field offense. In each trial,

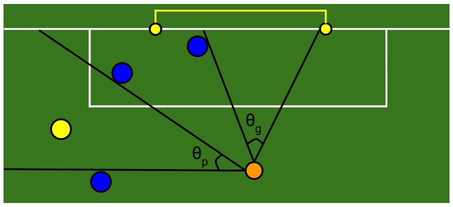

Figure 2: Open angle from ball to the goal avoiding the blue goalie and the open angle from the ball to the yellow teammate.

the agent is placed on a team randomly selected from the 7 teams described in Section 5.2. Performance is measured by the fraction of the time that the resulting team scores.

We compare several strategies for selecting from the policies learned by playing with previously encountered teammates. The performance is bounded above by the Correct Policy line, where the agent knows its teammate's behavior type and therefore which policy to use. The lower bound on performance is given by the Random Policy line, where the agent randomly selects which policy to use. The Combined Policy line shows the performance if the agent learns a single policy using the data collected from all possible teammates. Combined Policy is an intuitive baseline because it represents the view of treating the problem as a single agent learning problem, where the agent ignores the differences between its teammates. Finally, we additionally consider the team's performance without the ad hoc agent, denoted Down One. This baseline shows the intuitive metric of whether the ad hoc agent helps the team.

We then compare two more intelligent methods for selecting models, as described in Section 4.2. Specifically, our agent must decide which of the 7 policies to follow as it does not know its new teammate's behavior type. The Bandit line represents PLASTIC–Policy that uses an ϵ-greedy bandit algorithm to select policies. Other bandit algorithms were tested as were other values of ϵ, but ϵ-greedy with $\epsilon = 0.1$ decreasing to 0 outperformed these other methods. The PLASTIC–Policy line shows the performance our approach, using loss-bounded Bayesian updates to maintain probabilities over which previously learned policy to use. Specifically, it is the performance when using $\eta = 0.1$ and modeling noise with normal distributions with $\sigma = 4.0$ for distance differences and $\sigma = 40°$ for orientation differences.

6.1 Limited Half Field Offense

Our first set of results are in the limited version of the HFO game, involving 2 offensive players versus 2 defenders (including the goalie). Therefore, the agent only needs to adapt to a single teammate. This limited version of the problem reduces the number of state features to 8 and the number of actions while holding the ball to 4, while the number of actions away from the ball stays at 7. The results are shown in Figure 3, with the error bars showing the standard error.

The difference between the Correct Policy and Random Policy lines shows that selecting the correct policy to use is important for the agent to adapt to its teammates. The gap between the Correct Policy and Combined Policy shows than knowing the correct teammate is better that grouping all teammates together. The Bandit line looks like it is not learning, but over 100 episodes, it does improve its performance to 0.377. Its slow speed is due to the fact that its observations are noisy estimates of the policies' effectiveness and are only received after each game of HFO. The Down One line is not shown, but has a constant performance of 0.190, as playing down a player is a significant handicap.

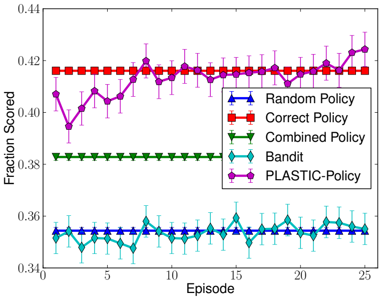

Figure 3: Scoring frequency in the limited half field offense task.

On the other hand, the PLASTIC–Policy line shows fast improvement, converging to the performance of the Correct Policy line. This quick adaptation is due to two factors: 1) the better estimations of which policy fits the teammates and 2) the frequency of the updates. The estimations of the probabilities are better as they measure how each agent moves, rather than only using a noisy estimate of how the policy performs. The updates are performed after every action rather than after each episode; so updates are much more frequent. These two factors combine to result in fast adaptation to new teammates using PLASTIC–Policy. Using a two population binomial test with $p < 0.01$, PLASTIC–Policy significantly outperforms Combined Policy, Bandit, Random Policy, and Down One for all episodes in the graph.

6.2 Full Half Field Offense

Our second set of results are in the full HFO game with 4 offensive players versus 5 defenders (including the goalie). In this setting, our agent needs to adapt to its three teammates to score against the five defenders. The results for this setting shown in Figure 4. As in Section 6.1, the upper bound on performance is given by Correct Policy and the lower bound is given by Random Policy. The Bandit setting learns slowly, reaching a performance of 0.341 over 100 episodes. Down One performs poorly again, achieving a score of 0.290. Once again, PLASTIC–Policy quickly converges to the correct policy's performance, outperforming the Bandit and Combined lines. These results show that PLASTIC–Policy quickly learns to cooperate with unknown teammates. Using a two population binomial test with $p < 0.05$, PLASTIC–Policy outperforms Combined Policy following episode 3 and outperforms Bandit, Random Policy, and Down One with $p < 0.01$ for all episodes.

7 Conclusion

The majority of past research on ad hoc teamwork has focused on domains such as grid worlds or matrix games and

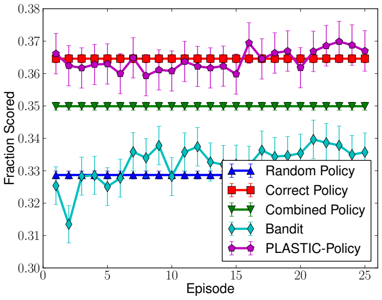

Figure 4: Scoring frequency in the full half field offense task.

in which only a few agents interact. However, the dream of applying ad hoc teamwork techniques to real world problems requires scaling these methods up to more complex and noisy domains that involve many agents interacting. This work presents a step towards this goal by introducing the PLASTIC–Policy algorithm to enable ad hoc teamwork in these types of domains. PLASTIC–Policy learns policies to cooperate with past teammates and then reuses these policies to cooperate with new teammates. PLASTIC–Policy is tested in the simulated robot soccer domain, the most complex domain that such an algorithm has been evaluated on.

This paper represents a move towards scaling up reasoning about ad hoc teamwork to complex domains involving many agents. One area for future work is applying this approach to real robots; for example, in future drop-in player challenges run at the RoboCup competition, such as those held in 2013 and 2014 (MacAlpine et al. 2014). Another interesting avenue for future research is reasoning about agents that are learning about the ad hoc agent over time.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287) and ONR (21C184-01).

References

Akiyama, H. 2010. Agent2d base code release. http://sourceforge.jp/projects/rctools.

Albus, J. S. 1971. A theory of cerebellar function. *Mathematical Biosciences* 10(12):25 – 61.

Albus, J. S. 1975. A new approach to manipulator control cerebellar model articulation control (cmac). *Transactions on ASME, J. of Dynamic Systems, Measurement, and Control* 97(9):220–227.

Barrett, S.; Stone, P.; Kraus, S.; and Rosenfeld, A. 2013. Teamwork with limited knowledge of teammates. In *AAAI*.

Blum, A., and Mansour, Y. 2007. *Algorithmic Game Theory*. Cambridge University Press. chapter Learning, regret minimization, and equilibria.

Bowling, M., and McCracken, P. 2005. Coordination and adaptation in impromptu teams. In *Proceedings of the Nineteenth Conference on Artificial Intelligence (AAAI)*, 53–58.

Brafman, R. I., and Tennenholtz, M. 1996. On partially controlled multi-agent systems. *Journal of Artificial Intelligence Research (JAIR)* 4:477–507.

Conitzer, V., and Sandholm, T. 2007. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning (MLJ)* 67.

Decker, K. S., and Lesser, V. R. 1995. Designing a family of coordination algorithms. In *International Conference on Multi-Agent Systems (ICMAS)*, 73–80.

Deisenroth, M. P.; Neumann, G.; and Peters, J. 2013. A survey on policy search for robotics. *Foundations and Trends in Robotics* 2(1-2):1–142.

Doshi, P., and Zeng, Y. 2009. Improved approximation of interactive dynamic influence diagrams using discriminative model updates. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. In *Journal of Machine Learning Research (JMLR)*, 503–556.

Gal, Y., and Pfeffer, A. 2008. Network of influence diagrams: Reasoning about agents' beliefs and decision-making processes. *Journal of Artificial Intelligence Research (JAIR)* 33:109–147.

Gmytrasiewicz, P. J., and Doshi, P. 2005. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research (JAIR)* 24(1):49–79.

Grosz, B., and Kraus, S. 1996a. Collaborative plans for complex group actions. *Artificial Intelligence (AIJ)* 86:269–368.

Grosz, B. J., and Kraus, S. 1996b. Collaborative plans for complex group action. *Artificial Intelligence (AIJ)* 86(2):269–357.

Horling, B.; Lesser, V.; Vincent, R.; Wagner, T.; Raja, A.; Zhang, S.; Decker, K.; and Garvey, A. 1999. The TAEMS White Paper.

Jones, E.; Browning, B.; Dias, M. B.; Argall, B.; Veloso, M. M.; and Stentz, A. T. 2006. Dynamically formed heterogeneous robot teams performing tightly-coordinated tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 570 – 575.

Kalyanakrishnan, S.; Liu, Y.; and Stone, P. 2007. Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*. Berlin: Springer Verlag. 72–85.

Liemhetcharat, S., and Veloso, M. 2012. Modeling and learning synergy for team formation with heterogeneous agents. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

MacAlpine, P.; Genter, K.; Barrett, S.; and Stone, P. 2014. The RoboCup 2013 drop-in player challenges: Experiments in ad hoc teamwork. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Stone, P.; Kaminka, G. A.; Kraus, S.; and Rosenschein, J. S. 2010. Ad hoc autonomous agent teams: Collaboration without precoordination. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)* 7:83–124.

Valtazanos, A., and Ramamoorthy, S. 2013. Bayesian interaction shaping: Learning to influence strategic interactions in mixed robotic domains. In *Proceedings of the Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 63–70.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College, Cambridge, UK.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for ad hoc autonomous agent teams. In *The 22th International Joint Conference on Artificial Intelligence (IJCAI)*.